

TI II

Sommersemester 2010

PD Dr. Katinka Wolter

4. Aufgabenblatt

Ausgabe Abgabe
31.05.10 11.06.10

Kontakt bei Fragen

Matthias Dräger, Viet Do, Marco Jeschke, Uwe Kuehn, Markus Rudolph
{mdraeger/do/kuehn/mjeschke/rudolph}@mi.fu-berlin.de

Problem 1: Pipelinetakte (2+2+2+4+2=12)

Gehen Sie im Folgenden von einer einfachen 5-stufigen Pipeline aus: Befehl holen (IF), Befehl dekodieren (ID), Operanden holen (OF), Ausführung (EX), Rückspeichern (WB). Weiterhin liegt eine reine Load/Store-Architektur ohne architekturelle Beschleunigungsmaßnahmen (z.B. forwarding, reordering etc.) oder Hardware zur Erkennung von Hemmnissen vor. Operanden können erst dann aus Registern geholt werden, nachdem sie zurück gespeichert wurden.

ADD X, Y, Z steht für $Z := X + Y$.

- Ist die Befehlsfolge
ADD R1, R2, R3
ADD R3, R5, R4
ohne das Einfügen von NOPs ausführbar ohne das Konflikte entstehen?
- Ist die Befehlsfolge
ADD R1, R2, R3
NOP
ADD R3, R5, R4
ohne das Einfügen von NOPs ausführbar ohne das Konflikte entstehen?
- Wie viele Takte dauert die vollständige Abarbeitung der folgenden Befehlsfolge? Fügen Sie unter Umständen eine geeignete Anzahl NOPs ein, damit die Folge bearbeitet werden kann.
 $Z := X + Y$
 $A := Z + B$
- Wie lange (in Takten) dauert die vollständige Abarbeitung der folgenden Befehlsfolge?
 $X := Z * Y$
 $A := X * B$
 $D := X + B$
- Wie groß ist im Idealfall die Beschleunigung der Programmabarbeitung mit dieser Pipeline?

Problem 2: Sprungvorhersage (3)

Betrachten wir 3 Vorhersagemethoden:

- Sprünge werden nie ausgeführt
- Sprünge werden immer ausgeführt
- Dynamische Vorhersage mit einer Vorhersagegenauigkeit im Durchschnitt von 95 %

Nehmen wir an, bei diesen Methoden entstehen keine Kosten, wenn die Vorhersage stimmt, und es entstehen Kosten in Höhe von 3 Zyklen, wenn die Vorhersage nicht stimmt.

Welches Verfahren eignet sich für die folgenden Sprünge am besten?

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

- Ein Sprung, der mit einer Häufigkeit von 10% ausgeführt wird.
- Ein Sprung, der mit einer Häufigkeit von 96% ausgeführt wird.
- Ein Sprung, der mit einer Häufigkeit von 70% ausgeführt wird.

Problem 3: dynamische Sprungvorhersage (1+2+2+6=11)

- Warum verwendet man eine dynamische Sprungvorhersage?
- Wie funktioniert ein 1-Bit-Predictor bei der dynamischen Sprungvorhersage?
- Was ist beim 2-Bit-Predictor anders? Für welche Fälle ist er besser geeignet?
- Wie häufig liegen bei den folgenden Fällen der 1-Bit-Predictor und der 2-Bit-Predictor(Hysteresis Scheme) statistisch betrachtet richtig bei ihrer Vorhersage? Gehen Sie von einem großen Wert von n aus, sowie für eine Wahrscheinlichkeit von 75%, dass p wahr ist. Beide Prädiktoren starten dabei im Zustand „Predict Strongly Taken“.

```
1) for i=1:n {  
    ...  
}  
2) for i=1:n {  
    for j=1:n {  
        ...  
    }  
}  
3) for i=1:n {  
    if p {  
        ...  
    }  
}
```

Problem 4: MMIX – Berechnung einer Quersumme (3+3+3+3=12)

Schreiben Sie ein Programm, das den Benutzer auffordert eine beliebige Zahl mit maximal 10 Stellen einzugeben. Von der eingegebenen Zahl soll die Quersumme¹ berechnet und ausgegeben werden.

Das Hauptprogramm lässt sich in vier Teile zerlegen:

1. Eingabe des Nutzers über die Standard-Eingabe.
2. Konvertierung der Eingabe in eine Zahl.
3. Berechnung der Quersumme dieser Zahl.
4. Ausgabe der Quersumme.

Implementieren Sie alle Programmteile als Unterprogramme und rufen sie diese aus dem Hauptprogramm auf!

¹Die Quersumme einer natürlichen Zahl bezeichnet die Summe aller Ziffern der gegebenen Zahl.
Beispiel: Die Quersumme von 654871 ist $6+5+4+8+7+1 = 31$

TI II

Sommersemester 2010
PD Dr. Katinka Wolter

Problem 5: Assembler-Kontrollstrukturen und Bubblesort (2+2+2+4=10)

Geben Sie MMIX Assembler-Befehle zur Simulation folgender Kontrollstrukturen an:

- a) while $X > Y$ do S;
- b) if $A = B$ then begin $X:=X+1$; $Y:=Z$ end else $A:=B$;
- c) for $J:= \text{LAST}$ downto FIRST do S;
- d) Verwenden Sie a) – c) zur MMIX Assemblerprogrammierung des nachfolgenden sog. Bubblesort-Algorithmus. Alle Zahlen seien vom Datentyp *wyde*.

```
last:=num;
while last > 0 do begin
  pairs:=last-1;
  last:=0;
  for j:=1 to pairs do
    if list[j] > list[j+1] then begin
      tmp:=list[j];
      list[j]:=list[j+1];
      list[j+1]:=tmp;
      last:=j;
    end;
  end;
end;
```